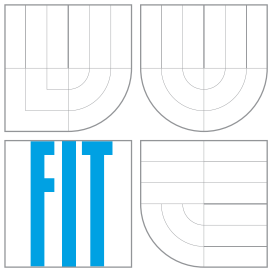
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KNIHOVNA PRO VIZUALIZACI LOGICKÝCH OBVODŮ

LIBRARY FOR VISUALIZATION OF DIGITAL CIRCUITS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

WALTER SCHERFEL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK VAŠÍČEK, Ph.D.

BRNO 2016

Abstrakt

Táto bakalárska práca sa zaoberá štúdiou automatického generovania schém a následne návrhom a implementáciou knižnice na automatickú vizualizáciu logických obvodov. Knižnica je implementovaná v jazyku C++ s využitím knižníc Qt (na vizualizáciu) a Boost (na prácu s XML štruktúrou a parsovanie vstupu). Výsledky tejto práce umožňujú, narozdiel od väčšiny ostatných nástrojov plniacich podobný účel, užívateľovi použiť knižnicu, implementovanú v tejto práci, vrámci svojho projektu a prípadne ju doplniť vlastnou implementáciou.

Abstract

This bachelor thesis is devoted to the automated schematic generation. The primary goal is to design and implement a library for visualization of digital circuits. The library is implemented in C++ language and depends on a popular Boost (used for work with XML structure and parsing input) and Qt (used for visualization) library. The library is designed in such a way that it can be easily embedded into a user application.

Kľúčové slová

automatický generátor schém, ASG, C++, Qt, logické obvody, schéma obvodu

Keywords

automatic scheme generator, ASG, C++, Qt, digital circuits, circuit scheme

Citácia

SCHERFEL, Walter. *Knihovna pro vizualizaci logických obvodů*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Vašíček Zdeněk.

Knihovna pro vizualizaci logických obvodů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Zdeňka Vašíčka, Ph.D. .

Uviedol som všetky literárne pramene a publikácie, zo ktorých som čerpal.

.....
Walter Scherfel
18.5.2016

Poďakovanie

Ďakujem Ing. Zdeňkovi Vašíčkovi, Ph.D. za vedenie mojej bakalárskej práce.

Obsah

Kapitola 1

Úvod

Elektronické obvody tvoria základný kameň všetkej vytváranej elektroniky. Pomocou nich sa dajú reprezentovať všetky komponenty počítača, monitoru a pod. Návrh zložitého elektronického obvodu je väčšinou proces, do ktorého je zapojených niekoľko ľudí. Títo ľudia musia medzi sebou komunikovať. Tu nastáva problém reprezentácie elektronického obvodu. Človeku je väčšinou bližšia reprezentácia grafická, než textová alebo formálna. Grafická reprezentácia by mala byť jednoducho pochopiteľná a mala by byť využiteľná napríklad na uľahčenie komunikácie v rámci tímu. Na tento účel je možné použiť nástroje typu Automatický generátor schém (Automated Scheme Generator [1]). Cieľom tejto práce je implementovať knižnicu, ktorá bude spĺňať práve tento účel.

Pri čítaní tohoto dokumentu je predpokladaná základná znalosť problematiky logických obvodov. Táto bakalárska práca implementuje knižnicu pre načítanie a zobrazovanie logických obvodov. Treba podotknúť, že sa v tejto práci nezaujímame logikou, funkčnosťou a použitím spracovávaných obvodov.

Text tejto technickej správy je rozdelený do šiestich kapitol. Po úvode nasleduje prehľad problematiky, ktorý uvedie termíny a pojmy, s ktorými budú pracovať ďalšie kapitoly. Ďalšia kapitola sa zaoberá návrhom knižnice, identifikovaním problémov a návrhom ich riešenia. V nasledujúcej kapitole je popísaná sada navrhnutých funkcií implementovaných v knižnici. Nadvädzujúca kapitola rozoberá implementáciu a algoritmy použité v rámci knižnice. Predposledná kapitola je venovaná vyhodnoteniu výsledkov práce pomocou spracovania niekoľkých obvodov. Záver je venovaný zhrnutiu dosiahnutých výsledkov a prínosu práce.

Kapitola 2

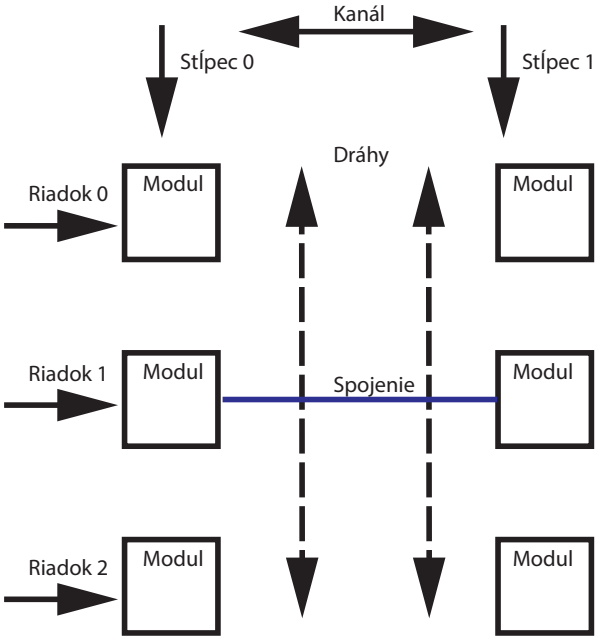
Prehľad problematiky

Táto kapitola popisuje termíny súvisiace s automatickým generovaním schém a problémy, ktoré musí generátor schém vyriešiť, aby dosiahol grafického výstupu. Vstupom automatického generátoru schém implementovaného v tejto práci je popis obvodu pomocou netlistu, výstupom je grafická schéma obvodu.

2.1 Termíny

Nasledujúce termíny sú často používané v spojení s logickými obvodmi a ich generovaním:

- logický obvod - obvod zložený z logických hradiel a spojení medzi nimi s jedným smerom toku dát,
- schéma obvodu - grafická reprezentácia obvodu v 2D (dvojdimenzionálnom) priestore, zväčša na obrazovke,
- modul - jedna zložka obvodu, ktorá má vstupy, výstupy, môže to byť hradlo, vstup alebo výstup obvodu,
- hradlo - modul, spĺňajúci logickú funkciu(AND, OR, NOT a pod.),
- spojenie - fyzické spojenie dvoch hradiel,
- netlist - popis obvodu ktorý popisuje aké moduly sa nachádzajú v obvode a ako sú medzi sebou prepojené,
- nástroj typu ASG (Automatic Schematic Generator [1]) - automatický generátor schém, je to nástroj na vytváranie schém elektronických obvodov z popisu(zväčša netlistu),
- polohovanie hradla - určovanie riadku a stĺpcu jednému určitému hradlu,
- polohovanie spojenia - určovanie pozície spojenia v rámci schémy jednému určitému spojeniu dvoch hradiel,
- kanál - miesto medzi dvomi stĺpcami hradiel,
- dráha - vertikálne miesto v rámci kanálu, ktoré využívajú spojenia.



Obr. 2.1: Grafická ukážka niekoľkých pojmov.

Na obrázku 2.1 je vysvetlených niekoľko pojmov definovaných v zozname v rámci tejto sekcie.

Jednou z možných reprezentácií popisu elektronických obvodov je netlist, ktorý je popísaný v ďalšej sekcii.

2.2 Netlist

Netlist je textový alebo binárny popis elektronického obvodu. V netliste je špecifikovaný zoznam hradiel, ktoré sa vyskytujú v obvode, pri čom tiež určuje ako sú medzi sebou tieto hradlá prepojené.

Vstupom pre nástroje typu ASG je väčšinou práve tento spôsob popisu elektronického obvodu. Pre tento projekt relevantný formát netlistu špecifikuje nasledujúce údaje:

- vstupy, výstupy obvodu a ich označenie,
- logické hradlá v rámci obvodu a ich typ (logické funkcie AND, OR a pod),
- spôsob, akým sú hradlá prepojené medzi sebou, a akým sú pripojené na vstupy a výstupy obvodu.

Dôvodom rozšírenosti netlistov je ich malá veľkosť a jednoduchá automatická spracovateľnosť. Nie sú však dobre čitateľné pre ľudského pozorovateľa, preto vzniká potreba pre nástroje typu ASG. Tie transformujú netlist do grafickej reprezentácie, ktorá je popísaná v ďalšej sekcii.

2.3 Grafická schéma obvodu

Grafická schéma obvodu je grafickou reprezentáciou obvodu. Táto grafická reprezentácia nie vždy presne odpovedá reálne vyrobenému obvodu. Zachováva však jeho logickú funkčnosť a

prepojenia. Grafická schéma obvodu sa používa hlavne kvôli jej prehľadnosti a jednoduchšej pochopiteľnosti pre ľudského pozorovateľa.

Študovaním grafických schém obvodu sa dá odvodiť niekoľko opakujúcich sa charakteristík:

- tok dát vždy vedie jedným smerom (väčšinou zľava doprava),
- skladá sa z dvoch základných častí, tými sú moduly a spojenia medzi nimi,
- všetky spojenia sa skladajú z horizontálnych a vertikálnych čiar na seba navzájom kolmých,
- moduly sú umiestnené v mriežke tzn. majú polohu popísateľnú číselným označením stĺpca a riadku,
- vstupy sú všetky umiestnené v rámci schémy v stĺpci úplne naľavo,
- výstupy sú všetky umiestnené v rámci schémy v stĺpci úplne napravo,
- moduly sa nemôžu medzi sebou prekrývať.

Pre prehľadnosť schémy je vhodné tieto charakteristiky zachovať.

Existuje niekoľko kritérií, podľa ktorých sa posudzuje prehľadnosť obvodu. Medzi používané patria napríklad počet krížení spojení, počet zahnutí spojení a celková dĺžka spojení.

2.4 Základné podproblémy vytvárania schémy obvodu

Podproblémy, ktoré by mal ASG vyriešiť sa dajú rozdeliť podľa dvoch kritérií.

Prvé kritérium, je či sa zaoberáme polohou modulov a spojení z grafického alebo logického hľadiska. Výstupom logického polohovania modulov a spojení by mala byť ich logická poloha vo výslednom grafe, tzn. riadok a stĺpec pre moduly, dráha a body zalomenia pre spojenia. Výstupom grafického polohovania modulov a spojení by mala byť ich geometrická pozícia, tzn. na ktorých pixeloch začína obrázok modulu, na ktorom pixely začína a končí spojenie a pod. .

Druhé kritérium je či sa zaoberáme určovaním pozície modulov, alebo či sa zaoberáme určovaním pozície spojení. Výstupom určovania pozície modulov by malo záležať na type určovania. Buď to je buď logická pozícia (stĺpec, riadok) alebo geometrická pozícia (presný bod kde bude vykreslená značka modulu). Výstup určovania pozície spojení by mal byť podobný výstupu určovania pozície modulov.

Skombinovaním týchto dvoch kritérií dostaneme štyri základné podproblémy, ktoré by mal nástroj typu ASG vyriešiť:

- určenie logickej pozície modulov,
- určenie logickej pozície spojení,
- určenie geometrickej pozície modulov,
- určenie geometrickej pozície spojení.

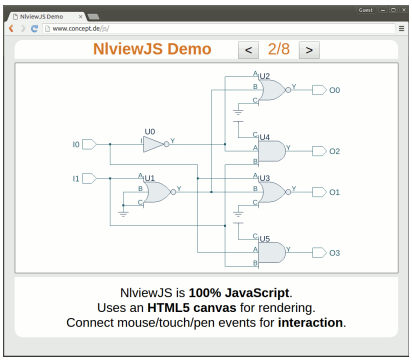
V sekciách, vnorených v kapitole 3.1, sú navrhnuté algoritmické riešenia týchto podproblémov.

2.5 Komerčne dostupné riešenia

Vzhľadom na to, že návrh elektronických obvodov je nutný takmer vo všetkých odvetviach vývoja elektroniky, už v minulosti existoval dopyt po nástrojoch typu ASG. Jedným z takýchto riešení je NLview

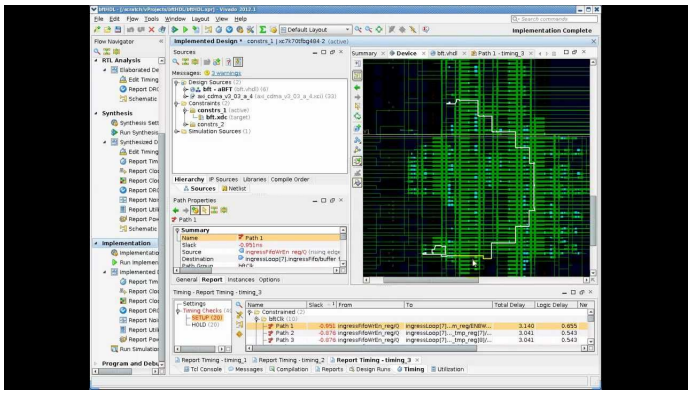
NLview [3] implementuje komponent na zobrazovanie elektronických obvodov. Tento komponent je dostupný v niekoľkých užívateľských rozhraniach, ako napríklad Qt, Java, wxWidges, Javascript a iné. Výhodou použitia komponentu NLview je možnosť použiť ju v niekoľkých jazykoch, GUI a pod. Toto riešenie v dnešnej dobe adoptovala rada predných dodávateľov CAD softwaru v oblasti navrhu hardware, ako je napríklad Xilinx [13] alebo Altera [6].

V obrázku 2.2 je ukážka z demo aplikácie dostupnej na stránkach NLview [3].



Obr. 2.2: Ukážka výstupu knižnice NLview.

V rámci softwaru Xilinx je použitý produkt Vivado. Vivado [4] je nástroj na návrh, simuláciu, verifikáciu a prácu s logickými obvodmi. Je súčasťou systému Xilinx, ktorý je používaný napríklad na návrh FPGA čipov. Používa už zmienenú komponentu NLview, ktorá je popísaná v predchádzajúcej sekcii. V obrázku 2.3 je ukážka nástroju Vivado.



Obr. 2.3: Ukážka programu Vivado.

Kapitola 3

Návrh knižnice na vizualizáciu logických obvodov

Táto kapitola sa zaoberá určením problémov, ktoré musí knižnica implementujúca ASG vyriešiť. Následne navrhuje možné riešenia zmienených problémov.

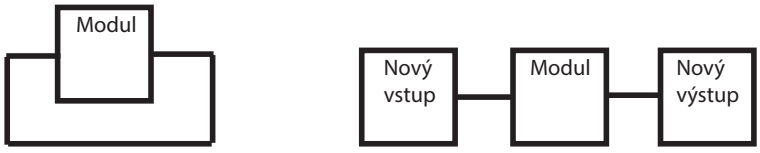
3.1 Návrh riešenia podproblémov vytvárania schémy obvodu

V tejto sekcii si navrhneme riešenia základných podproblémov zmienených v 2.4 .

Pri čítaní tejto sekcie si treba uvedomiť, že ideálnou schémou by bola schéma, kde sa nevyskytujú kríženia ani zalomenia spojení, spojenia majú minimálnu možnú dĺžku a podobne. Dosiahnuť týchto kritérií pri reálnych obvodoch v zobrazovaní na 2D priestore je veľakrát nemožné. Algoritmy, ktoré sa zaoberajú riešením tohoto podproblému, sa snažia čo najbližšie priblížiť ideálnej schéme.

3.1.1 Určovanie logickej pozície modulov

Táto fáza by mala začínať odstránením spätných väzieb. Tento krok môže byť vykonaný pomocou pridania nového vstup a výstupu obvodu. Vstupy a výstupy, ktoré sú pridané v tejto fáze algoritmu sa nazývajú interné vstupy a výstupy. Následne je na vstup hradla na miesto, kde pred tým bola spätná väzba, privedená hodnota interného vstupu a výstup hradla je vyvedený do interného výstupu (viď obrázok 3.1). Vďaka tomuto kroku je dosiahnutý rovnaká funkcionalita ako spätná väzba. Výhodou je, že program môže s takto transformovanými spätnými väzbami pracovať veľmi podobne, ako s ostatnými spojeniami, ktoré spájajú hradlá medzi sebou. Pre prehľadnosť je vhodné takto pridané vstupy a výstupy nejakým spôsobom označiť.



(a) Stav pred transformáciou spätnej väzby.

(b) Stav po transformácií spätnej väzby.

Obr. 3.1: Ukážka transformácie spätnej väzby.

Nasledovať by malo určenie stĺpca, kde bude modul umiestnený. Tu by mal platiť vzťah, že číselné poradie stĺpca, v ktorom bude hradlo umiestnené, musí byť väčšie ako najväčšia hodnota číselného poradia stĺpca spomedzi jeho vstupov. Zároveň musí byť toto číselné poradie menšie ako najmenšia hodnota číselného poradia stĺpca spomedzi jeho výstupov. Najjednoduchší prístup je umiestniť hradlo hneď za jeho vstupy. Môže ale nastať situácia, že hradlo je možné umiestniť do niekoľkých stĺpcov. Napríklad v obvode je hradlo, ktoré má vstupy v stĺpcoch 2 a 3, pričom výstupy má až v stĺpcoch 7 a 8, potom pre je toto hradlo možné umiestniť do stĺpcov 4, 5 a 6. Záleží na konkrétnej implementácií či túto možnosť vezme do úvahy a porovná pomocou heuristík (počet hradiel v stĺpci a pod.) a vyberie najoptimálnejšiu pozíciu alebo, či hradlo jednoducho umiestni v stĺpci danom najvyššou hodnotou stĺpca spomedzi vstupov hradla.

Po určení stĺpca by mal ASG určiť riadok, v ktorom bude dané hradlo umiestnené. Existuje viacero spôsobov na určenie riadku. Algoritmy, pomocou ktorých sa určuje riadok hradla sa snažia minimalizovať zložitosť výslednej schémy. Väčšinou sa ako kritéria zložitosti berú údaje, ako napríklad počet krížení spojení, počet zahnutí spojení alebo celková dĺžka spojení. Riešenie tohoto podproblému býva časovo náročné, pretože pre N prvkov existuje celkom N! možností usporiadaní prvkov.

3.1.2 Určovanie logickej pozície spojení

Určenie logickej pozície spojenia by malo pozostávať z priradenia dráhy (číslované zľava doprava) každému spojeniu, ktorého aspoň časť musí byť vertikálna. Toto priradenie by malo prebehnúť tak, aby sa spojenia neprekrývali. Prekrývanie spojení vedie k zvýšenej miere nejednoznačnosti a neprehľadnosti schémy obvodu. Je možné, že vzniknú kríženia spojov, ale cieľom algoritmu, ktorý rieši tento podproblém by malo byť mimo iné aj minimalizovanie počtu krížení spojení.

3.1.3 Určovanie geometrickej pozície modulov

Presné súradnice jednotlivých modulov by mali byť založené na polohe vypočítanej v logickej fáze. Vzdialenosti medzi stĺpcami modulov sa môžu líšiť, kvôli rôznemu počtu dráh medzi stĺpcami, pričom vzdialenosti medzi riadkami modulov sa môžu líšiť kvôli rôznemu počtu spojení prechádzajúcich okolo modulov.

3.1.4 Určovanie geometrickej pozície spojení

Posledný podproblém je vypočítanie presných súradníc spojení. Toto by malo zahŕňať výpočet súradníc začiatku spojenia, prípadných zahnutí spojenia a ukončenia spojenia.

Po ukončení určovania geometrickej pozície spojení by mal byť výsledok zobrazený do súboru alebo na obrazovku.

3.2 Výber algoritmu

V sekcii 2.4 sú popísané štyri algoritmické riešenia podproblémov, z ktorých by sa mal skladať základný algoritmus vizualizácie nielen logických obvodov. V sekcii 3.1 sú navrhnuté riešenia zmienených podproblémov. Tieto algoritmy sa ale dajú zoradiť niekoľkými spôsobmi.

Jedným z nich je určiť logické pozície modulov, následne ich geometrické pozície. Až potom začne algoritmus brať do úvahy spojenia medzi modulmi a určuje logické pozície spojení, a nasledovne ich geometrické pozície.

Ďalšou možnosťou je najprv si určiť logické pozície modulov a následne spojení, a až potom ich geometrické pozície.

V tejto práci je implementovaný druhý spôsob, kvôli viacerým možnostiam použiteľnosti knižnice. Vďaka tomu, že je implementovaný práve druhý zmienený spôsob, môže prípadný užívateľ po ukončení logickej časti výpočtu použiť výsledok na zobrazovanie pomocou iných algoritmov.

3.3 Návrh rozdelenia práce na funkčné celky

Vzhľadom na to, že v minulej sekcii bol vybraný algoritmus, ktorý najprv určuje logické pozície modulov a spojení a až potom určuje ich geometrické pozície, je vhodné rozdeliť program na dva funkčné celky.

Prvý bude určovať logické pozície modulov a spojení. Druhý celok bude určovať ich geometrické pozície a bude zaobstarávať zobrazovanie, a prípadné exportovanie grafických výsledkov.

3.4 Vstup knižnice

V prehľade problematiky je uvedené, že vstupom nástrojov typu ASG je väčšinou popis elektronického obvodu vo forme netlistu. Na fakulte, kde je táto práca vytváraná, sa používa formát netlistov .chr (a jeho binárna forma .bchr) . Preto je v tomto projekte použitý práve tento formát.

3.4.1 Formát chr(bchr)

Tento formát popisuje textovou alebo binárnou formou elektronický obvod. Textové popisy obvodu bývajú uložené v súboroch s koncovkou .chr (binárne popisy obvodu bývajú uložené v súboroch s koncovkou .bchr). Textový popis je pre človeka lepšie zrozumiteľným variantom popisu obvodu, preto budeme popisovať práve jeho štruktúru.

Popis obvodu začína znakom reťazcom `#file` po ktorom nasleduje názov obvodu. Ďalší riadok začína značkou `#%i` ktorá znamená, že nasledovať budú označenia vstupov oddelené čiarkami. Je treba podotknúť, že vstupy sú číslované implicitne od 0. Prvý vstup má označenie 0, druhý 1 atď. Toto číslovanie nie je v tomto formáte uvedené, ale je používané pri určovaní vstupov hradiel v nasledujúcej časti. Na ďalšom riadku sú podobným spôsobom označené výstupy, značka na začiatku riadku je však `#%o` . Nasledujúci riadok začína značkou `#%f` a sú v ňom čiarkami oddelené názvy logických funkcií. Logické funkcie sú tiež implicitne číslované od 0, podobne ako vstupy obvodu. V popise jednotlivých hradiel bude toto číselné označenie značené `ID_of_type`. Na ďalšom riadku nasledujú parametre obvodu vo forme číselných údajov v zložených zátvorkách. Významy jednotlivých po sebe nasledujúcich hodnôt sú nasledovné:

```
{num_of_inputs, num_of_outputs, num_of_rows, num_of_cols, block_inputs,
block_outputs, not_used}
```

- hodnota `num_of_inputs` určuje počet vstupov obvodu,

- hodnota `num_of_outputs` určuje počet výstupov obvodu,
- hodnota `num_of_rows` určuje počet riadkov obvodu, pre implementáciu tejto knižnice je tento údaj zanedbaný, pretože počet riadkov vo výslednej schéme záleží na výsledku logického polohovania modulov,
- hodnota `num_of_cols` určuje počet stĺpcov obvodu, pre implementáciu tejto knižnice je tento údaj zanedbaný, pretože počet stĺpcov vo výslednej schéme záleží na výsledku logického polohovania modulov,
- hodnota `block_inputs` určuje počet vstupov hradiel použitých v tomto obvode,
- hodnota `block_outputs` určuje počet výstupov hradiel použitých v tomto obvode,
- hodnota `not_used` nie je používaná.

Nasleduje popis jednotlivých hradiel v nasledujúcej forme. Tento príklad platí pre n-vstupé hradlo:

```
([ID], input_id, ... ,input_id_n, ID_of_type)
```

- hodnota `ID` určuje číselné označenie hradla, číslovanie hradiel začína hneď po poslednom označení vstupu (napríklad obvod má posledný vstup s označením 9, prvé hradlo bude teda mať označenie 10),
- hodnoty `input_id, ..., input_id_n` určujú číselné označenia vstupov obvodu,
- hodnota `ID_of_type` určuje číselné označenie logickej funkcie hradla.

Po ukončení popisu hradiel nasleduje posledná časť popisu obvodu a to je zoznam výstupov logického obvodu popisovaného v danom súbore. Nasledujúci príklad platí pre obvod s N výstupmi:

```
(output_id, ..., output_id_n)
```

- hodnoty `output_id, ..., output_id_n` určujú číselné označenie hradliel, ktoré sú pripojené na výstupy obvodu.

V prílohách je príklad textového popisu jednoduchého obvodu.

3.5 Uloženie alebo exportovanie medzivýsledku

Nasledujúci podproblém je predanie, prípadne uloženie medzivýsledkov medzi logickým a grafickým polohovaním. Základnou podmienkou formátu, ktorý bude využívaný na tento účel, je, že sa musí dať jednoducho serializovať. Tento podproblém by sa dal vyriešiť niekoľkými formátmi súborov:

- binárny súbor - výber tejto varianty by znamenal jednoduchý automatický zápis a čítanie, ale výstup by nebol čitateľný okom bez externých nástrojov,
- textový súbor - predávanie dát pomocou textového súboru by znamenalo veľmi dobrú čitateľnosť a editovateľnosť pomocou bežných nástrojov.

Pre účely tejto knižnice bol vybraný textový súbor, vďaka jednoduchej serializácií. Následne bolo nutné vybrať z nasledujúcich variantov:

- vlastný formát - jednou z možností bolo vytvoriť si vlastný formát, nevýhodou by ale bolo, že by tento formát nebol dobre spracovateľný v rámci iných projektov, keďže by na neho neexistovali podporné knižnice (načítanie a zápis),
- XML (Extended Markup Language) - tento formát sa najviac hodí na spomínanú funkciu hlavne kvôli tomu, že bol vytvorený za účelom predávania dát medzi aplikáciami, prípadne ich časťami,
- JSON (JavaScript Object Notation) - hoci je tento formát v textovej podobe, znamenal by relatívne zlú čitateľnosť oproti formátu XML.

Pre tento účel bol vybraný formát XML aj z dôvodu, že je rozšírený a knižnice na parsovanie XML súborov existujú v takmer každom jazyku.

V jazyku C++ plní funkciu tejto knižnice Boost.

3.5.1 Knižnica Boost

Knižnica Boost [5] je komunitou vytváraná multiplatformná knižnica, ktorá dopĺňa funkcionalitu štandardnej knižnice C++. Je vytváraná pre komerčné aj nekomerčné používanie. Kvalitu tejto knižnice potvrdzuje aj to, že jej niektoré časti sú, postupne v rámci vydávania nových štandardov(posledný je C++11), pridávané do štandardnej knižnice C++.

3.5.2 Význam jednotivých elementov výstupnej štruktúry XML

V tejto sekcii sú popísané jednotlivé elementy výstupu logického polohovania. V prílohe je jednoduchý príklad XML štruktúry, ktorý bol vytvorený knižnicou implementovanou v tejto práci. V nasledujúcich sekciách sú popísané všetky elementy výslednej štruktúry.

Koreňový element Scheme

Koreňovým elementom výstupnej XML štruktúry je element `Scheme`. Vnorené elementy sú nasledujúce:

- element `Name` má v sebe uložené meno schémy,
- element `Rows_only_for_conns` uchováva indexy riadkov, ktoré sú určené iba na zobrazovanie spojení, (vnorené elementy sú popísané v podsekcií 3.5.2),
- element `Columns` má v sebe údaje o všetkých stĺpcoch, Údaje sú rozdelené po stĺpcoch, uložené v jednotlivých elementoch `Column` (štruktúra každého takého elementuje je popísaná v 3.5.2).

Element Rows only for conns

V tomto elemente sú uložené indexy riadkov, ktoré boli do obvodu pridané iba kvôli výskytu spojení, ktoré siahajú cez niekoľko kanálov. Tento údaj môže byť užitočný pre zobrazovaciu časť, pretože takéto riadky nepotrebujú byť tak široké ako riadky, kde sú uložené moduly. Ak sa ale užívateľ rozhodne naimplementovať si vlastné zobrazovanie, tento údaj vôbec nemusí použiť.

V elemente popisovanom v tejto sekcii je vnorený zoznam elementov typu `Index`. Elementy typu `Index` obsahujú vždy jediný vnorený element typu `Value`, v ktorom je číselná hodnota riadku, ktorý je použitý iba na zobrazovanie spojení. Číslovanie riadkov je vzostupné a začína od 0.

Element Column

V tomto elemente sú uložené informácie o jednom stĺpci modulov v rámci obvodu.

Elementy vnorené v elemente `Column` sú nasledujúce:

- element `Index` obsahuje číselné poradie stĺpca. Stĺpce sú, podobne ako riadky, číslované vzostupne od 0,
- element `Num_of_tracks` obsahuje počet dráh, ktoré sú využívané v rámci kanálu, ktorý je medzi momentálne spracovávaným a nasledujúcim stĺpcom,
- element `Nodes` má v sebe zoznam elementov typu `Node`. (štruktúra týchto elemntov je popísaná v 3.5.2).

Element Node

V tomto elemente sú uložené informácie o jednom module spolu so spojeniami, ktoré z daného modulu vedú.

Elementy vnorené v tomto elemente sú nasledujúce:

- element `Description` obsahuje popis modulu, používa sa iba pre označenie vstupov a výstupov obvodu,
- element `Row` obsahuje číselné poradie riadku v ktorom je modul umiestnený,
- element `Type` obsahuje číselné označenie typu modulu. Význam hodnôt týchto označení je popísaný nižšie v tejto sekcii,
- element `Number_of_outputs` obsahuje počet výstupov daného modulu,
- element `Outputs` je zoznam výstupných spojení daného modulu, jednotlivé spojenia sú popísané vo vnorených elementoch typu `Output` (štruktúra týchto elemntov je popísaná v 3.5.2).

Možné číselné hodnoty označenia typu modulu uloženého v elemente `Type` sú nasledujúce.

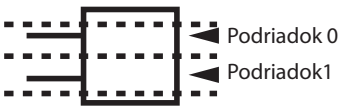
- 0 - prázdny modul, nemá sa nijakým spôsobom vykreslovať,
- 1 - vstup obvodu,
- 2 - výstup obvodu,
- 3 - hradlo typu And,
- 4 - hradlo typu NAnd,
- 5 - hradlo typu And s negovaným prvým vstupom,

- 6 - hradlo typu And s negovaným druhým vstupom,
- 7 - hradlo typu Or,
- 8 - hradlo typu NOr,
- 9 - hradlo typu Or s negovaným prvým vstupom,
- 10 - hradlo typu Or s negovaným druhým vstupom,
- 11 - invertor prvého vstupu
- 12 - invertor druhého vstupu,
- 13 - hradlo typu Xor,
- 14 - hradlo typu XNor,
- 15 - hradlo s výstupom, ktorý bude vždy obsahovať hodnotu 1,
- 16 - hradlo s výstupom, ktorý bude vždy obsahovať hodnotu 0,
- 17 - hradlo, ktorého výsledná hodnota bude vždy obsahovať prvý vstup hradla,
- 18 - hradlo, ktorého výsledná hodnota bude vždy obsahovať druhý vstup hradla.

Element Output

Tento element obsahuje informácie o jednom spojení vedenom z modulu, ktorý je jeho nadradeným elementom.

V nasledujúcom obrázku je graficky vysvetlený pojem podriadok, ktorý bude používaný v rámci ukladania údajov o spojení.



Obr. 3.2: Vysvetlenie pojmu podriadok.

Možné vnorené elementy sú nasledovné:

- element `Subrow_start` je číselné označenie podriadku (viď obrázok) v ktorom spojenie začína,
- element `Row_end` je číselné označenie riadku, v ktorom spojenie končí,
- element `Subrow_end` je číselné označenie podriadku v ktorom spojenie končí,
- element `Num_of_dest_inputs` je počet vstupov modulu, v ktorom spojenie končí,
- element `Track` je dráha použitá pre dané spojenie, ak dráha nie je pre dané spojenie potrebná, v tomto elemente je uložená hodnota -1.

3.6 Zobrazenie výsledkov

Ďalším podproblémom nad ktorým sa treba zamyslieť je zobrazenie výsledkov. Výstup z knižnice je nutné nejakým pre oko pochopiteľným spôsobom zobraziť, najlepšie pomocou GUI (grafické užívateľské rozhranie - graphics user interface). Užívateľ GUI obvykle ovláda pomocou ovládacích prvkov, ako napríklad roletka, tlačidlo a pod. Na účel jednoduchého vytvárania a manipulácie s GUI vzniklo veľa knižníc pre rôzne jazyky s rôznymi zameraniami. Známymi príkladmi pre C++ sú Qt a GTK+.

V zadaní tejto práce je uvedený implementačný jazyk C++ s využitím knižnice Qt. Preto sa táto práca inými možnosťami nezaoberá.

3.6.1 Knižnica Qt

Knižnica Qt[9] je knižnica na užívateľské rozhrania pre programy implementované v jazykoch C++ a Python. Táto knižnica má niekoľko vlastností, vďaka ktorým je veľmi používaná:

- prináša so sebou veľké množstvo už naimplementovaných tried, ktoré môže užívateľ knižnice ďalej rozširovať (vlastná implementácia interakcie s myšou a pod.),
- je natívne multiplatformná (Windows, Unix, Android a iné),
- rozširuje C++ o niektoré užitočné mechanizmy, najznámejšie z nich sú signály a sloty (citeqtsignals, cyklus foreach,
- grafické užívateľské rozhranie je možné vytvárať dvomi spôsobmi. Jeden je s použitím nástroja dizajnér v rámci nástroja QtCreator [8], druhý spôsob je s použitím modulu QtQuick [10],
- má automatickú správu pamäte vo forme hierarchie vytváraných objektov (poznámka týmto sa nemyslí dedičnosť), funguje to tak, že každý objekt má svojho rodiča, ak je jeho rodič zničený, je zničený aj samotný objekt a všetci jeho potomkovia,
- podporuje jednoduché vytváranie grafického výstupu pomocou Graphics View Frameworku [7], ten umožňuje zobrazovať text, vektorové obrázky, rastrové obrázky, čiary, vytvárať skupiny objektov, zisťovať kolízie a podobne.

3.7 Export výsledkov

Vlastnosťou, ktorú by budúci užívateľ knižnice mohol oceniť je možnosť exportovať výslednú schému ako obrázok pre ďalšie použitie.

Možné formáty sú nasledujúce:

- PNG(Portable Network Graphics) - export výsledkov pomocou PNG by pre užívateľa mohol znamenať nutnosť konvertovať výsledok do iného formátu, keďže PNG nie je tak rozšírený ako iné formáty obrázkov,
- JPG(Joint Photographic Experts Group) - tento formát síce znamená do istej miery stratu kvality, ale je jedným z najviac rozšírených formátov obrázkov na internete.

Pre túto prácu boli vybrané formáty PNG a JPG z dôvodu ich rozšírenosti.

Kapitola 4

Rozhranie knižnice

V tejto kapitole je popísané rozhranie a popis použitia knižnice, implementovanej v tejto práci.

4.1 Popis použitia knižnice

Knižnica implementovaná v tejto práci sa dá použiť niekoľkými spôsobmi. Jedným z nich je použiť ju iba na načítanie a výpočet logických polôh modulov a spojení. Výsledky tejto fáze sú dostupné vo forme XML, užívateľ na ne môže nadviazať vlastnou implementáciou grafického polohovania a samotného zobrazenia schémy. Výhodou môže byť napríklad to, že užívateľ môže výsledky vo forme XML použiť aj v programe v inom programovacom jazyku, v inom štýle zobrazovania, v inom GUI a podobne.

Ďalším zo spôsobov, ako použiť knižnicu implementovanú v tejto práci je použiť jazyk C++ s využitím Qt a použiť knižnicu na riešenie všetkých podproblémov ASG. V tomto prípade sa knižnica postará o celý proces zobrazenia schémy pre daný obvod. Výhodou tohto spôsobu jeho jednoduchosť za predpokladu, že užívateľ vo svojom projekte tiež použije C++ s využitím Qt. V tomto prípade je postup použitia nasledujúci. Vo svojej aplikácií si užívateľ vytvorí, buď v kóde alebo v dizajnéri v rámci nástroja QCreator, inštanciu `QGraphicsView`. Následne jej pomocou `QGraphicsView::setScene()` priradí inštanciu triedy `Scheme_gen_scene`, implementovanej v tejto knižnici, ako zobrazovanú scénu. Vďaka rozsiahlej implementácií `QGraphicsView` by užívateľ nemal mať nutnosť zaoberať sa vecami ako napríklad automatické prekreslenie scény pri pridaní prvku, zobrazenie a správne nastavenie roletiek (anglicky scrollbar) a podobne.

Pri používaním tejto knižnice je nutné mať na danom systéme nainštalovanú knižnicu Boost (viď sekcia 3.5.1).

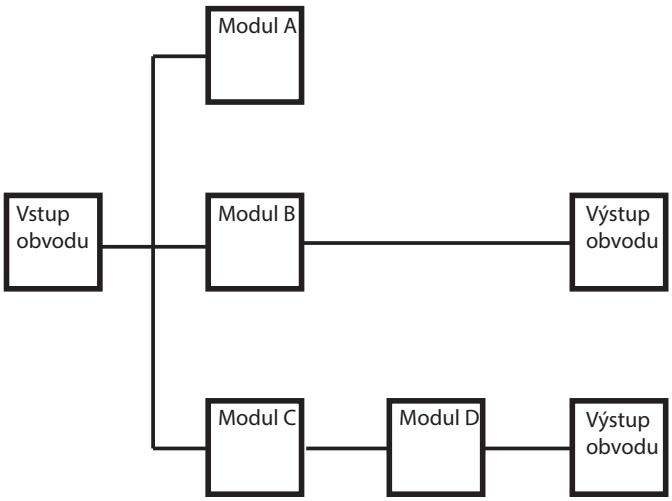
4.2 Rozhranie knižnice bez grafického zobrazovania v Qt

Rozhranie prvej časti knižnice, ktorá sa zaoberá logickým polohovaním modulov a spojení, je implementované v triede `Scheme_gen_logic`. Základné rozhranie prístupné užívateľovi popísané v nasledujúcich odstavcoch.

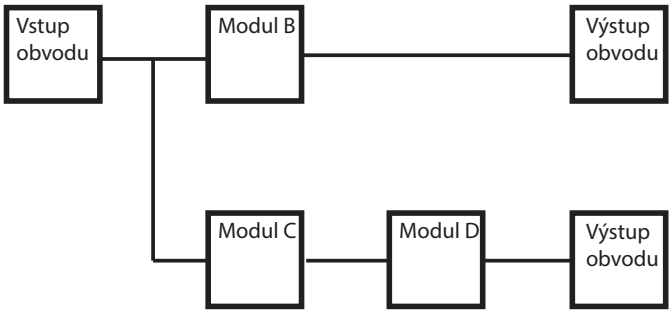
Metóda `create_scheme(std::string netlist, std::string name)` vytvorí schému z netlistu zadaným parametrom. Meno zadané parametrom sa používa pri exporte výsledkov do súboru, ktorý je popísaný nižšie. Táto metóda nastaví svojmu objektu vnútorné

premenné, ktoré reprezentujú moduly a spojenia tak, že výsledok je možné z neho pomocou ďalej popísaných metód získať alebo exportovať. Metóda `get_result()` slúži na získanie vytvorenej XML štruktúry obsahujúcej informácie o spracovanom obvode. Túto metódu je nutné volať až po volaní metódy `create_scheme()`. Návratovou hodnotou je XML štruktúra, ktorá je uložená v objekte typu `boost::property_tree::ptree`. Metóda `export_result()` je metóda, ktorá slúži na exportovanie vytvorenej XML štruktúry do XML súboru. Takto vyexportované súbory majú názov podľa parametru predanému metóde `create_scheme()` s koncovkou `.xml`. V súbore je následne uložená štruktúra popísaná v 3.5.2. Takto vyexportované súbory sú uložené do adresára `exported schemes`, ktorý knižnica vytvorí v adresári preloženej aplikácie, ktorá používa knižnicu implementovanú v tejto bakalárskej práci. Metóda `export_result(std::string filename)` je podobná `export_result()`, s tým rozdielom, že táto metóda dostane parametrom meno a cestu k súboru kde má XML štruktúru uložiť. Metóda `reset()` je metóda, ktorá vymaže momentálny uložený stav modulov a spojení a uvoľní pamäť používanú na uchovávanie údajov o momentálnom obvode. Volanie tejto funkcie pripraví objekt na spracovanie ďalšieho obvodu. Je vždy automaticky volaná na začiatku `create_scheme()`, aby sa premazali momentálne uložené moduly a spojenia a nekolidovali s novo načítaným obvodom. Metóda `omit_not_used_nodes(bool omit)` nastaví hodnotu vnútornej premennej `do_not_calc_not_used_nodes`. Hodnota tejto premennej sa testuje pri načítaní obvodu. Ak je v tejto premennej uložená hodnota `true`, tak sa po načítaní z obvodu vymažú moduly, ktoré nie sú napojené, buď priamo, alebo nepriamo (cez ich výstupné moduly) na aspoň jeden z výstupov obvodu (viď obrázky 4.1 a 4.2). Ak je v tejto premennej uložená hodnota `false`, mazanie týchto modulov neprebehne. Ak užívateľ túto hodnotu nenastaví, pri načítaní sa premažú moduly nepripojené aspoň na jeden z výstupov obvodu.

V obrázku 4.1 sú ukázané moduly v rámci obvodu. Modul A nie je pripojený na výstup, takže je možné, že v rámci načítania bude tento modul vymazaný z obvodu. Modul B je priamo pripojený na jeden z výstupov obvodu. Modul C je pripojený na výstup nepriamo pomocou modulu D, na ktorý je pripojený priamo. Modul D je pripojený priamo na jeden z výstupov obvodu. Po vymazaní modulov nepripojených aspoň na jeden z výstupov obvodu bude obvod vyzerať ako v obrázku 4.2.



Obr. 4.1: Ukážka hradla nepripojeného na výstup.



Obr. 4.2: Obvod z obrázku 4.1 po odstránení modulov nepripojených na výstup.

4.3 Rozhranie knižnice s grafickým zobrazovaním v Qt

Nasledujúce metódy rozhrania sú implementované v rámci triedy `Scheme_gen_scene`:

- metóda `create_scheme(std::string netlist, std::string name)` je metóda, ktorá vytvorí a zobrazí schému zo zadaného súboru alebo reťazca,
- metóda `create_scheme(QFile netlist, std::string name)` spĺňa veľmi podobnú úlohu, ako metóda popísaná v predchádzajúcom bode, s tým rozdielom, že v tomto prípade je vstup uložený v súbore, nie v reťazci,
- metóda `omit_not_used_nodes(bool omit)` nastaví zobrazovanie modulov, ktoré nie sú pripojené priamo, alebo nepriamo (viď obrázok) na aspoň jeden z výstupov,
- metódy `export_to_jpg(QString filename)` a `export_to_png(QString filename)` exportujú momentálne zobrazenú schému do JPG alebo PNG súboru s menom uvedeným v parametre.

Vďaka tomu, že `Scheme_gen_scene` je potomkom `QGraphicsScene` má užívateľ tejto triedy možnosť používať metódy implementované v knižnici Qt. Za zmienku stojí najmä `QGraphicsScene::clear()`, ktorá vymaže všetky momentálne zobrazené prvky zo scény.

Kapitola 5

Implementácia knižnice na vizualizáciu logických obvodov

Knižnica implementovaná v tejto bakalárskej práci je rozdelená, ako je popísané v kapitole 3.3, na dva funkčné celky. Prvá časť rieši logické polohovanie modulov a spojení, ktorého základný algoritmus je rozpísaní v kapitole 3.1.1 a 3.1.2 , druhá rieši ich geometrické polohovanie, ktorého základný algoritmus je rozpísaný v kapitole 3.1.3 a 3.1.4 . Druhá časť knižnice sa zaoberá aj finálnym zobrazením schémy na obrazovku alebo do súboru.

Táto kapitola sa venuje vnútornej implementácií obidvoch spomenutých častí.

5.1 Logické polohovanie

Logika tejto časti knižnice je implementovaná v triede `Scheme_gen_logic`, ktorej rozhranie je popísané v kapitole 4.2 . Pomocnými triedami sú `Scheme_gen_node` a `Scheme_gen_conn`. `Scheme_gen_node` slúži na uchovávanie dát o jednom module v rámci obvodu. `Scheme_gen_conn` splňuje podobnú úlohu, ale pre spojenia v rámci obvodu.

5.1.1 Použité prostredie

Pre túto časť knižnice je použité C++ s využitím knižnice Boost. Je možné využiť tu aj knižnicu Qt, ale v rámci návrhu bola vybraná varianta bez použitia Qt, kvôli lepšej znovupoužiteľnosti. Ak by tam bola použitá aj knižnica Qt, tak by prípadný užívateľ knižnice, implementovanej v tejto práci, musel pridať do svojho projektu aj knižnicu Qt. V momentálnom stave tak nemusí učiniť. Môže jednoducho vyexportovať XML s výsledkami logického polohovania a tie použiť na zobrazovanie v užívateľskom rozhraní používajúcom aj iné knižnice pre GUI (GTK [11], WPF [12] a pod.) .

5.1.2 Popis štruktúr a metód

Vnútorné dátové štruktúry triedy `Scheme_gen_logic` sú nasledujúce:

- premenná `node_map` v sebe uchováva informácie o moduloch, vo forme inštancií objektov triedy `Scheme_gen_node`, v momentálne spracovávanom obvode,
- premenná `conn_list` v sebe uchováva informácie o spojeniach, vo forme inštancií objektov triedy `Scheme_gen_conn`, v momentálne spracovávanom obvode,

- v premennej `num_of_tracks` je pre všetky kanály uložené koľko dráh má určitý kanál,
- v premennej `rows_only_for_conns` sú uložené indexy riadkov, ktoré boli do obvodu pridané, aby nimi boli vedené spojenia, ktoré siahajú cez viacero kanálov.

Metódy, ktoré používajú vyššie uvedené dátové štruktúry sú nasledujúce:

- metóda `load_netlist()` načíta netlist daný parametrom a naplní `node_map` modulmy daného obvodu,
- metóda `resolve_node_position()` sa stará o logické polohovanie modulov a zápis výsledku do `node_map` (podrobný postup je uvedený nižšie v **??**),
- metóda `resolve_connection_position()` sa stará o logické polohovanie spojení a zápis výsledku do `conn_list` (podrobný postup je opäť uvedený nižšie v **??**),
- metóda `resolve_feedback_loops()` transformuje prípadnú spätnú väzbu na nový výstup a nový vstup, tak ako je popísané v kapitole 3.1.1,
- metóda `make_space_for_conns()` vytvorí riadky navyše pre spojenia, ktoré siahajú cez viacero kanálov, indexy takto vytvorených riadkov zapíše do
- V `rows_only_for_conns`,
- metóda `find_track_for_conn()` nájde vhodnú dráhu pre spojenie, atribúty spojenia ako je hlavne riadok začiatku vertikálnej časti a riadok konca vertikálnej časti musia byť zadané v parametroch.

Logika tejto časti knižnice je implementovaná v triede `Scheme_gen_logic`, ktorej rozhranie je popísané v kapitole 4.3 .

5.1.3 Použitý algoritmus

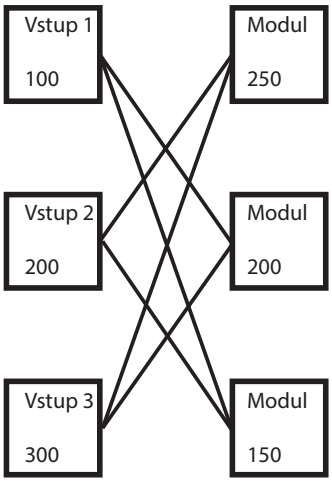
Spracovanie obvodu začína tým, že používateľ tejto časti knižnice (pozn. ktorým je buď druhá časť knižnice venujúca sa zobrazovaniu alebo program naimplementovaný užívateľom, ktorý sa rozhodol si zobrazovaciu časť implementovať sám) zavolá metódu `create_scheme()` nad inštanciou triedy `Scheme_gen_logic`. V tejto metóde je implementovaný celý postup. V ďalšej časti je tento postup krok po kroku popísaný.

Najprv sa pomocou metódy `load_netlist()` načíta netlist a vytvorené moduly sa uložia do inštančnej premennej `node_map`. Vzhľadom na to, že vo formáte .chr, popísanom v 3.4.1, sú hodnoty v rámci popisu modulov oddelené čiarkami, na rozdelenie načítaného reťazca na jednotlivé hodnoty je v tejto časti implementácie použitý `boost::tokenizer`. Ten dokáže podľa zadaného znaku rozdeliť reťazec na časti. Následne, ako je už popísané v 4.2, ak je nastavená premenná `do_not_calc_not_used_nodes` na hodnotu `true`, tak sú z obvodu po načítaní vymazané moduly, ktoré nie sú či už priamo alebo nepriamo pripojené na aspoň jeden z výstupov obvodu. Nasleduje začiatok logického polohovania modulov.

Algoritmus výpočtu logických polôh modulov

Algoritmus na výpočet logickej pozície modulov sa dá rozdeliť na dve časti. Prvou je vypočítať stĺpec, do ktorého sa má modul zaradiť. Tento výpočet prebieha ešte v rámci funkcie `load_netlist()`. Algoritmus prejde každý modul, ktorý je v obvode. Ak to nie je vstup alebo výstup obvodu, tak program zistí najväčší index stĺpca spomedzi vstupov daného modulu, a modul priradí do nasledujúceho stĺpca. V prípade, že modul je vstup, tak modul zaradí do najľavejšieho stĺpca (s indexom 0), ak je to výstup tak tento modul zaradí do najľavejšieho stĺpca. Následne sa vykoná transformácia spätných väzieb pomocou volania metódy `resolve_feedback_loops()`. Postup a výsledky tohoto kroku sú popísané v kapitole 3.1.1. Nasleduje posledná časť výpočtu logických polôh modulov, a tou je výpočet riadku, do ktorého sa má modul zaradiť. Priebeh tohoto výpočtu je nasledovný [2] . Jedná sa o algoritmus založený na propagácií hodnôt. Každému vstupu obvodu sa priradí bublinková hodnota (anglicky bubble value) podľa číselného poradia riadku v ktorom sú umiestnené. Ostatným modulom a výstupom sa priradí hodnota rovná priemeru bublinkových hodnôt jeho vstupu. Priraďovanie hodnôt prebieha v metóde `set_bubble_values()`. Nasledovne sú moduly v rámci stĺpca zoradené podľa vypočítaných hodnôt vzostupne. Tento algoritmus je zameraný hlavne na elimináciu križení spojení.

V obrázku 5.1 je ukázaný obvod s nastavenými bublinkovými hodnotami. V obrázku 5.2 sú už moduly zoradené v rámci stĺpca podľa ich bublinkových hodnôt.

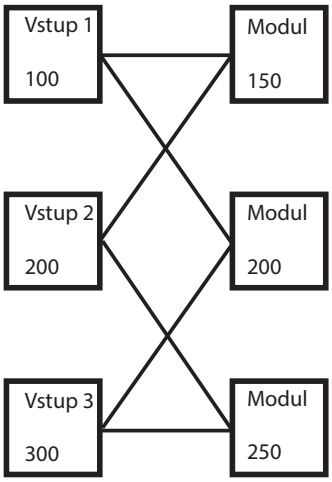


Obr. 5.1: Obvod s nastavenými bublinkovými hodnotami pred radením.

Algoritmus výpočtu logických polôh spojení

Po výpočte logických polôh modulov nasleduje výpočet logických polôh spojení.

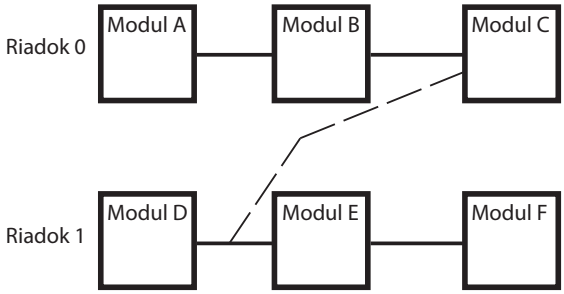
Najprv si algoritmus vytvorí dátovú štruktúru pre dráhy, ktorú bude používať ďalej. Nasleduje vytváranie miesta pre spojenia, ktoré siahajú cez viacero kanálov. Toto zaobstaráva funkcia `make_space_for_conns()`. Táto funkcia vytvorí riadok modulov s typom `Empty` (pozn. možné typy modulov sú definované pri definícií triedy `Scheme_gen_node`), ktoré sú následne vsadené do riadku, ktorým bude spojenie povedené. Tieto moduly medzi sebou funkcia spojí tak, aby zaistili vytvorenie požadovaného spojenia (viď obrázok 5.3 a 5.4). Tento krok zaistí, že v nasledujúcom výpočte logických polôh spojení nie je nutné spracovávať spojenia, ktoré siahajú cez viacero kanálov, pretože boli týmto krokom rozkúskované



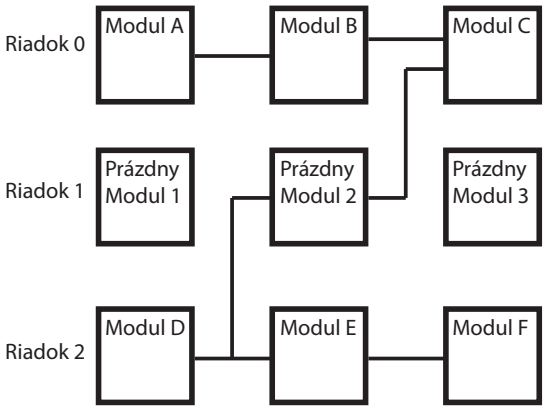
Obr. 5.2: Obvod s nastavenými bublinkovými hodnotami po radení.

na spojenia siahajúce iba cez jeden kanál.

Na nasledujúcich obrázkoch je ukázané pridávanie riadkov. Modul D je spojený s modulom C. Toto spojenie siaha cez niekoľko kanálov. Pôvodný stav je v obrázku 5.3 . V 5.4 je stav po pridaní riadku pomocných modulov, pri čom číslo riadku 1 sa pridá do `rows_only_for_conns`.



Obr. 5.3: Pôvodný stav.

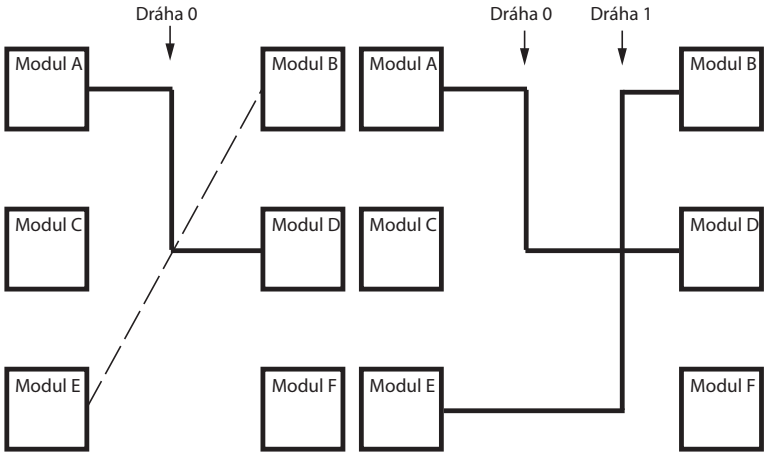


Obr. 5.4: Stav po pridaní pomocného riadku.

Potom začne cyklus, ktorý postupne prechádza stĺpce modulov a následne každý modul

v `node_map`. Tento cyklus spracováva stĺpce aj samotné moduly v rámci stĺpca vo vzostupnom poradí. Najprv zoradí výstupy momentálne spracovávaného modulu podľa riadku vzostupne. Nasleduje prechádzanie výstupov modulu. Pre každý výstup sa vytvorí jedna inštancia objektu `Scheme_gen_conn`, ktorej sa nastavia správne hodnoty(riadok začiatku a konca spojenia a pod.) a uloží sa do inštančnej premennej `conn_list`, ktorá sa používa pri exporte výsledkov do XML štruktúry. Ak majú spojované moduly iné hodnoty indexu riadku, znamená to, že spojenie potrebuje aj vertikálnu časť, tým pádom potrebuje mať určenú dráhu (pozn. anglicky track). Dráhu spojeniu určí funkcia `find_track_for_conn()`, ktorá zistí či by sa momentálne spracovávané spojenie nedalo umiestniť do jednej z už existujúcich dráh. Ak ani v jednej nie je pre spracovávané spojenie miesto, vytvorí sa nová dráha a spojenie tam zaberie jednotlivé časti dráhy (viď obrázok 5.5). Po spracovaní všetkých spojení v rámci jedného kanálu funkcia uloží počet dráh do inštančnej premennej `num_of_tracks`.

Na nasledujúcich obrázkoch je ukázané pridávanie dráh. Modul E je spojený s modulom B. V dráhe 0 pre toto spojenie nie je miesto, takže je vytvorená nová dráha. Pôvodný stav je v obrázku 5.5 v ľavej časti. V 5.5, v pravej časti, je stav po pridaní dráhy.



(a) Stav pred pridaním dráhy. (b) Po pridaní dráhy.

Obr. 5.5: Pôvodný stav

5.2 Geometrické polohovanie a zobrazovanie výsledkov

Geometrické polohovanie a konečné vykreslenie obvodu je implementované v triede `Scheme_gen_scheme`, ktorá je potomkom triedy `QGraphicsScene` implementovanej v knižnici Qt. Jej rozhranie je popísané v sekcii 4.3 .

5.2.1 Popis štruktúr a metód

V tejto triede sú použité nasledujúce dátové štruktúry:

- premenná `num_of_tracks` zastáva rovnakú úlohu ako v prvej časti knižnice (popísané v 5.1.2),
- premenná `rows_only_for_conns` tiež zastáva rovnakú úlohu ako v prvej časti knižnice (popísané v 5.1.2),

- premenná `logic` je samotná inštancia triedy `Scheme_gen_logic` ktorá sa stará o výpočet logických pozícií. (rozhranie tohoto objektu je popísané v 4.2 a použítý algoritmus je popísaný v 5.1.3) ,
- premenná `node_size` určuje veľkosť textúry pre moduly,
- v premennej `node_textures` sú uložené textúry používané pri zobrazovaní modulov.

Metódy používajúce tieto dátové štruktúry sú nasledovné:

- metóda `display_results()` vykreslí obvod zadaný XML štruktúrou, predanou parametrom, do scény,
- metóda `calc_node_position()` vypočíta a vráti pozíciu ľavého horného rohu modulu, ktorého číslo riadku a stĺpca je zadané parametrom,
- metóda `calc_vertical_position()` vypočíta vertikálnu pozíciu horného okraja modulu s číslom riadku zadaným v parametre,
- metóda `calc_horizontal_position()` vypočíta horizontálnu pozíciu ľavého okraja modulu s číslom stĺpca zadaným v parametre.

5.2.2 Použitý algoritmus

Spracovanie obvodu v tejto časti knižnice tiež začína tým, že užívateľ zavolá metódu `create_scheme()`. V tomto prípade užívateľ potrebuje použiť knižnicu Qt vo svojej implementácií.

V metóde `create_scheme()` sa zavolá rovnomenná metóda objektu logiky uloženom v inštančnej premennej `logic`, aby si objekt logiky vytvoril XML štruktúru popisujúcu logické pozície modulov a spojení v obvode, ktorého netlist bol zadaný parametrom. Následne je volaná metóda `display_results()` ktorá zobrazí výsledky získané z objektu logiky do scény.

Táto metóda si zo vstupnej XML štruktúry, popísanej v 3.5.2, najprv naplní hodnoty `rows_only_for_conns`. Následne prechádza stĺpce, ktoré sú umiestnené vzostupne v XML štruktúre. Najprv si pre každý stĺpec prečíta počet dráh v kanály, ktorý je medzi momentálne spracovávaným a nasledujúcim stĺpcom. Následne prechádza moduly v danom stĺpci, ktoré sú tiež vzostupne umiestnené v XML. Každý modul vykreslí do scény na súradnice, ktoré si vypočíta pomocou metódy `calc_node_position()` . V rámci modulu prechádza výstupné spojenia daného modulu, ktoré vykresľuje do scény. V prípade, že dané spojenie má dráhu nastavenú na -1, tak to znamená, že spojenie nepoužíva dráhu, tým pádom nemá vertikálnu časť. V tomto prípade použije už vypočítanú pozíciu momentálne spracovávaného modulu, pripočíta odsadenie, ktoré je vypočítané s pomocou hodnoty uvedenej v `Subrow_start` a vykreslí jednoduchú čiaru medzi vstupom a výstupom. V prípade, že spojenie využíva dráhu, tým pádom má vertikálnu časť, si algoritmus vypočíta súradnice zahnutí spojenia a vykreslí spojenie po častiach.

Vďaka návrhu štruktúry, popísanom v 3.5.2, je možné vykresľovať moduly a spojenia už behom načítania štruktúry. Z toho vyplýva, že načítanie tejto štruktúry je nutné vykonať iba jeden krát bez nutnosti opravy vykreslených prvkov.

Kapitola 6

Vyhodnotenie výsledkov práce

Táto kapitola sa zaoberá vyhodnotením výsledkov práce na niekoľkých logických obvodoch s rôznymi parametrami a veľkosťou.

V tejto práci je implementovaná knižnica na vizualizáciu logických obvodov. Časová náročnosť spracovania obvodu záleží jak na obvode samotnom, tak na algoritmoch použitých pri vytváraní výslednej schémy. V kapitole 5 je popísané, aké algoritmy sú použité v rámci implementácie tejto knižnice.

V rámci spracovania obvodu, ktoré je popísané v 5, sa niekoľko krát prechádzajú všetky moduly v rámci obvodu, prípadne ich výstupy. Vďaka využitiu algoritmu z dokumentu [2] sa program dokonca vyhne nutnosti porovnávať všetky možné zoradenia modulov v rámci stĺpca. To znamená, že moduly, prípadne ich výstupy sa vždy v rámci riešenia jedného podproblému prechádzajú iba raz. To znamená, že program by mal mať lineárnu zložitosť.

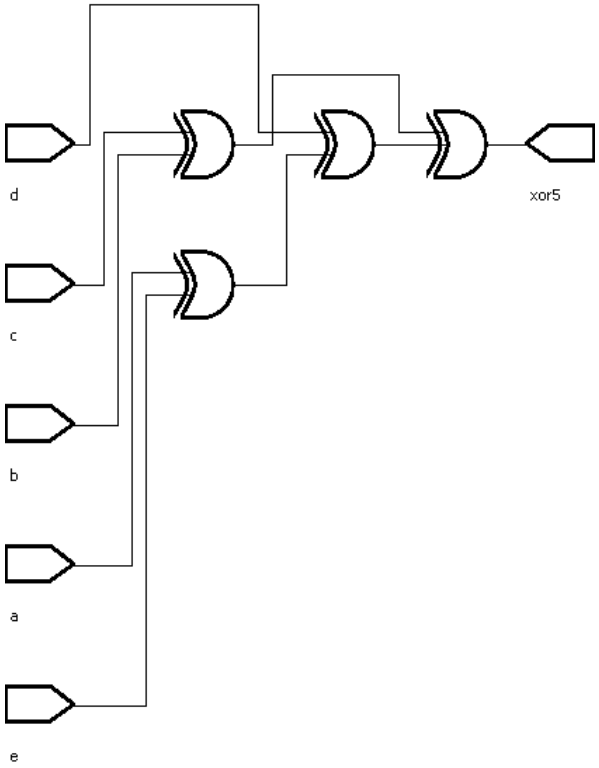
V nasledujúcej tabuľke sú uvedené časy trvania spracovania vybraných ôsmych obvodov. Počet vstupov logických hradiel je v prípade týchto obvodov vždy 2, počet výstupov logických hradiel je v prípade týchto obvodov vždy 1.

V tabuľke 6.1 sú uvedené časy trvania spracovania obvodov. Vyplýva z nich, že knižnica má zložitosť knižnice sa viac približuje kvadratickej ako lineárnej. Môže to byť zapríčinené prácou s vnútornými štruktúrami, väčšinou typu `std::vector`, s ktorými program pracuje s pomocou štandardnej knižnice C++. Na určenie presnej časovej zložitosti by bolo nutné získať hlbšie pochopenie spôsobu, akým štandardná knižnica C++ pracuje so všetkými dátovými štruktúrami, ktoré sú použité v knižnici, implementovanej v tejto práci (viď sekcie 5.1.2 a 5.2.1).

V obrázku 6.1 je ukázaná vizualizácia obvodu uloženom v súbore xor5.chr .

Tabuľka 6.1: Tabuľka trvania spracovania ôsmych obvodov.

Názov	Počet vstupov obvodu	Počet výstupov obvodu	Počet modulov	Spracovanie logickou časťou (ms)	Spracovanie grafickou časťou (ms)
xor5.chr	5	1	10	1	13
majority.chr	5	1	15	1	14
decod.chr	5	16	57	10	69
cu.chr	14	11	80	24	114
b12.chr	15	9	93	25	112
clip.chr	9	5	149	71	216
9symml.chr	9	1	230	147	356
ex5.chr	8	63	616	1222	1271



Obr. 6.1: Vizualizácia obvodu uloženom v súbore xor5.chr.

Kapitola 7

Záver

V tejto práci je implementovaná knižnica na vizualizáciu logických obvodov. Knižnica umožňuje použitie dvomi spôsobmi. Je možné ju použiť buď iba na čiastočný výpočet logickej fáze (viď sekcie 3.1.1 a 3.1.2), alebo ako celok. V rámci návrhu programu bola vytvorená XML štruktúra na zápis medzivýsledku (viď sekcie 3.5.2), na ktorú môže potom užívateľ knižnice nadviazať vlastnou implementáciou. Výhodou prvého spôsobu je možnosť implementovať projekt, ktorý využíva knižnicu implementovanú v tejto práci, bez využitia Qt. Možnými rozšíreniami knižnice by bola napríklad možnosť exportu do vektorového formátu, možnosť vytvorený obrázok editovať (myšou alebo v rámci kódu), podpora pre MOS tranzistory a podobne.

Keď sa spätne pozrieme na zadanie a jeho body, môžeme povedať, že bolo v tejto práci splnené. Zoznámenie s princípmi automatického vytvárania schém je popísané v kapitole Prehľad problematiky (viď kapitola 2). Sada funkcií na vytváranie schém je popísaná v kapitole Rozhranie (viď kapitola 4). Návrh tejto knižnice je popísaný v kapitole Návrh (viď kapitola 3), implementácia a konkrétne použité algoritmy sú popísané v kapitole Implementácia (viď kapitola 5). V kapitole Vyhodnotenie výsledkov práce (viď kapitola 6) je popísaná časová zložitosť vytvoreného riešenia.

Literatúra

[1] Lageweg, C.: *Designing an Automatic Schematic Generator for a Netlist Description*. 1998, [Online; navštívené 23.1.2016].
URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.6.3952&rep=rep1&type=pdf`

[2] T.M. Parng, Y.S. Jehng, L.G.Chen: *ASG: Automatic schematic generator*. 1991, [Online; navštívené 14.3.2016].
URL `http://www.sciencedirect.com/science/article/pii/0167926091900045`

[3] WWW stránky: *Products of Concept Engineering - overview on schematic generation and visualization*. 2016-05-06 [cit. 2016-03-04], [Online; navštívené 21.2.2016].
URL `http://concept.de/products.html`

[4] WWW stránky: *Vivado Design Suite*. [cit. 2016-03-05], [Online; navštívené 5.3.2016].
URL `http://www.xilinx.com/products/design-tools/vivado.html#overview`

[5] WWW stránky: *Boost C++ Libraries*. [cit 2016-03-23], [Online; navštívené 14.3.2016].
URL `http://www.boost.org/`

[6] WWW stránky: *FPGA, SoC and CPLD from Altera*. [cit. 2016-03-23], [Online; navštívené 5.3.2016].
URL `https://www.altera.com/`

[7] WWW stránky: *Graphics View Framework | Qt Widgets 5.6*. [cit. 2016-03-23], [Online; navštívené 22.3.2016].
URL `http://doc.qt.io/qt-5/graphicsview.html`

[8] WWW stránky: *Qt - Product | The IDE*. [cit. 2016-03-23], [Online; navštívené 22.3.2016].
URL `http://doc.qt.io/qt-5/graphicsview.html`

[9] WWW stránky: *Qt Project* . [cit. 2016-03-23], [Online; navštívené 22.3.2016].
URL `http://www.qt.io/`

[10] WWW stránky: *Qt Quick 5.6*. [cit. 2016-03-23], [Online; navštívené 22.3.2016].
URL `http://doc.qt.io/qt-5/qtquick-index.html`

[11] WWW stránky: *The GTK+ Project*. [cit. 2016-03-23], [Online; navštívené 23.3.2016].
URL `http://www.gtk.org/`

[12] WWW stránky: *Windows Presentation Foundation*. [cit. 2016-03-23], [Online; navštívené 23.3.2016].
URL `https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx`

[13] WWW stránky: *Xilinx*. [cit. 2016-03-23], [Online; navštívené 5.3.2016]. URL `http://www.xilinx.com/`

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Obsahom CD priloženom k tejto práci sú nasledovné veci:

- zdrojové súbory knižnice v priečinku src,
- technická správa ku knižnici v priečinku doc,
- príklady použitia knižnice v priečinku examples,
- výsledky práce vo forme vstupných obvodov a súborom s časmi spracovania spomínaných obvodov sú v priečinku results.

Príloha B

Príklad netlistu vo formáte chr

```
#file: twolexamples_abc\xor5.blif (linear)
#%i d,c,b,a,e
#%o xor5
#%f IDA,AND2,OR2,XOR2,INVA,NAND2,NOR2,XNOR2,ZERO,ONE
{5,1,4,1,2,1,4}([5]4,3,3)([6]2,1,3)([7]5,0,3)([8]6,7,3)([9]1,2,3)(8)
%\chapter{RelaxNG Schéma konfiguračního soboru}
%\chapter{Plakat}
```

Príloha C

Príklad výstupu z prvej časti knižnice vo formáte XML

```
<?xml version="1.0" encoding="utf-8"?>
<Scheme>
  <Name>first</Name>
  <Rows_only_for_conns/>
  <Columns>
    <Column>
      <Index>0</Index>
      <Num_of_tracks>0</Num_of_tracks>
      <Nodes>
        <Node>
          <Row>0</Row>
          <Type>1</Type>
          <Number_of_outputs>1</Number_of_outputs>
          <Outputs>
            <Output>
              <Subrow_start>0</Subrow_start>
              <Row_end>0</Row_end>
              <Subrow_end>0</Subrow_end>
              <Num_of_dest_inputs>1</Num_of_dest_inputs>
              <Track>-1</Track>
            </Output>
          </Outputs>
        </Node>
      </Nodes>
    </Column>
    <Column>
      <Index>1</Index>
      <Num_of_tracks>0</Num_of_tracks>
      <Nodes>
        <Node>
          <Row>0</Row>
          <Type>3</Type>
```

```
            <Number_of_outputs>1</Number_of_outputs>
            <Outputs>
              <Output>
                <Subrow_start>0</Subrow_start>
                <Row_end>0</Row_end>
                <Subrow_end>0</Subrow_end>
                <Num_of_dest_inputs>1</Num_of_dest_inputs>
                <Track>-1</Track>
              </Output>
            </Outputs>
          </Node>
        </Nodes>
      </Column>
      <Column>
        <Index>2</Index>
        <Num_of_tracks>0</Num_of_tracks>
        <Nodes>
          <Node>
            <Row>0</Row>
            <Type>2</Type>
            <Number_of_outputs>0</Number_of_outputs>
            <Outputs/>
          </Node>
        </Nodes>
      </Column>
    </Columns>
</Scheme>
```